CLAIMS

1. A method in a computer system for preparing a task to be swapped out from processor utilization, the computer system having multiple processors and an operating system, each processor having multiple streams for simultaneously executing threads of the task, the task having one or more teams of threads, each team representing threads executing on a single processor, the method comprising:

raising an exception for each stream of each processor currently executing a thread of the task; and

in response to the raising of the exception, for each stream executing a thread,

saving the state of the stream;

determining whether the stream is a team master stream;

when the stream is not the team master stream, quitting the stream;

when the stream is the team master stream,

waiting for all other streams executing threads in the same team to quit;

determining whether the stream is a task master stream,

when the stream is not the task master stream, notifying the operating system that the team for this processor is ready to be swapped out;

when the stream is the task master stream,

waiting for all other teams to notify the operating system that the team is ready to be swapped out; and

notifying the operating system that the task is ready to be swapped out.

2. A method in a computer system for preparing a task to be swapped out from processor utilization, the computer system having multiple processors and an operating system, each processor having multiple streams for executing threads of the task, the task having one or more teams of threads, each team representing threads executing on a single processor, the method comprising:

for each team, designating one stream that is executing a thread as a team master stream;

designating one stream that is executing a thread as a task master stream for the task;

for each team master stream, notifying the operating system that the team is ready to be swapped out when each other thread of the team has quit its stream; and

for the task master stream, notifying the operating system that the task is ready to be swapped when each of the other teams have notified the operating system that that team is ready to be swapped out.

3. The method of claim 2 wherein the operating system swaps out the task upon receiving the notification that the task is ready to be swapped out.

4. The method of claim 2 wherein each stream stores is own state before quitting the stream or notifying the operating system.

5. The method of claim 2 wherein each stream that is not a team master stream quits its stream.

6. The method of claim 2 wherein the notifying of the operating system by the task master stream includes indicating whether the task is blocked so

that the operating system can defer swapping in the task until an event occurs to unblock the task.

7. The method of claim 2 wherein each team master stream notifies the operating system of the number of streams that were executing threads so that the operating system can defer swapping in the task until enough streams are available to execute each of the threads that were executing when the task was swapped out.

8. A method in a computer system for preparing a task to be swapped out from processor utilization, the computer system having a processor and an operating system, the processor having multiple streams for executing threads of the task, the method comprising:

designating one stream that is executing a thread as a master stream;

saving the state of each stream that is executing a thread;

under control of each stream that is not the master stream, quitting the stream;

under control of the master stream, notifying the operating system that the task is ready to be swapped out.

9. The method of claim 8 wherein the operating system swaps out the task upon receiving the notification that the task is ready to be swapped out.

10. The method of claim 8 wherein each stream saves its own state.

11. The method of claim 8 including upon being swapped in:

starting execution of a master stream; and

under control of the master stream,

creating a stream corresponding to each stream that quit when the task was swapped out; and

initializing each created stream based on state information saved before the stream quit.

12. The method of claim 8 wherein the notifying of the operating system by the master stream includes indicating whether the task is blocked so that the operating system can defer swapping in the task until an event occurs to unblock the task.

13. The method of claim 8 wherein the task notifies the operating system of the number of streams that were executing threads so that the operating system can defer swapping in the task until enough streams are available to execute each of the threads that were executing when the task was swapped out.

14. A method in a computer system for scheduling tasks, the method comprising:

notifying a task that it is being preempted from processor utilization;

in response to the notification, receiving an indication from the task that it is ready to be swapped out and an indication as to whether the task is block;

when an indication that the task is blocked is received, deferring the swapping in of the task until an event occurs that may cause the task to become unblocked.

15. The method of claim 14 wherein the computer system is a multithreaded computer system.

16. The method of claim 14 wherein in response to the notification, the task saves its state.

17. A method in a computer system for restarting the execution of a task that has been swapped out of processor utilization, the method comprising:
starting execution of a master stream; and
under control of the master stream,
creating a stream corresponding to each stream that quit when the task was swapped out; and
initializing each created stream based on state information saved before the stream quit.

18. A method in a computer system for returning a stream to a task executing an operating system call that is blocked, the computer system having a processor with multiple streams, method comprising:
under control of the operating system executing on a stream, invoking a function provided by the task;
under control of the invoked function, executing instructions of the task on that stream; and
under control of the operating system, notifying the task when the operating system call is complete.

19. The method of claim 18 wherein the notifying includes
invoking a function provided by the task using a stream of the operating system; and
under control of that invoked function,
indicating that the operating system call is complete; and

invoking another operating system call to return the operating system stream to the operating system.

20. The method of claim 18 wherein the executing of instructions on that stream includes

indicating that a thread that invoked the operating system call is blocked; and

executing another thread on that stream.

21. A method in a computer system for performing an inter-thread long jump, the method comprising:

receiving an indication of a set jump location;

determining whether the set jump thread that set the set jump location is the same thread that is currently executing; and

when the set jump thread is not the same thread that is currently executing, setting the state of the set jump thread to execute a long jump indicating the set jump location.

22. The method of claim 21 wherein the setting of the state includes setting the program counter of the set jump thread to point to a long jump routine.

23. The method of claim 21 wherein when the set jump thread is blocked on an operating system call aborting the operating system call.

24. The method of claim 23 including starting the execution of the set jump thread.

25. The method of claim 21 wherein when the set jump thread is running notifying the set jump thread of the inter-thread long jump.

26. The method of claim 21 wherein when the set jump thread is running directing the set jump thread to enter a known state prior to setting the state of the set jump thread.

27. The method of claim 26 wherein the known state is a quiescent state.

28. The method of claim 21 including determining whether the set jump thread still exists.

29. The method of claim 21, 22, 23, 24, 25, 26, 27, or 28 wherein the computer system supports multiple streams.

30. A method in a computer system for a server to coordinate assignment of a resource to clients, the method comprising:

assigning the resource to a client;

receiving notification from the client assigned to the resource that the client is waiting for an occurrence of an event before the resource can be productively used;

upon receiving the notification, unassigning the resource from the client; and

after occurrence of the event, reassigning the resource to the client.

31. The method of claim 30 wherein the server upon receiving the notification assigns the resource to another client.

32. The method of claim 30 wherein the server is an operating system, the clients are tasks, and the resource is a processor resource of the computer system.

33. The method of claim 32 wherein the processor resource is single-threaded.

34. The method of claim 32 wherein the processor resource is multi-threaded.

35. The method of claim 32 wherein the operating system upon notification assigns the processor resource to another task.

36. The method of claim 32 wherein the task can only perform idle processing until the event occurs.

37. The method of claim 32 wherein the task can perform no processing until the event occurs.

38. The method of claim 32 wherein the operating system reassigns the processor resource to the task whenever an external event directed to the task occurs.

39. The method of claim 32 wherein the notification is received by the operating system in response to informing the task that the task is to be swapped out from processor resource utilization.

40. The method of claim 32 wherein the operating system assigns a task to the processor resource by assigning the task to a domain.

41. A method in a computer system for clients to coordinate assignment of a resource with a server, the method comprising:

determining whether the client cannot productively use the resource until an event occurs; and

when the client determines that it cannot productively use the resource, notifying the server that the client cannot productively use the resource

wherein the server unassigns the resource from the client until after an event occurs.

42. The method of claim 41 wherein the server is an operating system, the clients are tasks, and the resource is a processor resource of the computer system.

43. The method of claim 42 wherein the processor resource is single-threaded.

44. The method of claim 42 wherein the processor resource is multi-threaded.

45. The method of claim 42 wherein the operating system upon notification assigns the processor resource to another task.

46. The method of claim 42 wherein the task can only perform idle processing until the event occurs.

47. The method of claim 42 wherein the task determines whether it cannot productively use the processor resource until an event occurs after receiving an indication from the operating system that the task is to be swapped out from processor resource utilization.

48. The method of claim 42 wherein the task notifies the server after receiving an indication from the operating system that the task is to be swapped out from processor resource utilization.

49. The method of claim 42 wherein the task can perform no processing until the event occurs.

50. The method of claim 42 wherein the operating system assigns a task to the processor resource by assigning the task to a domain.

51. A computer-readable medium for causing an operating system executing on a computer to coordinate assignment of processor resources to task, by:

assigning a processor resource to a task, the task having productive instructions to execute using the assigned processor resource;

receiving notification from the task assigned to the processor resource that the task is waiting for an occurrence of an event before the task can continue execution of the productive instructions

upon receiving the notification, unassigning the processor recourse from the task; and

after occurrence of the event, reassigning the processor resource to the task.

52. The computer-readable medium of claim 51 wherein the operating system upon receiving the notification assigns the processor resource to another task.

53. The computer-readable medium of claim 51 wherein the processor is single-threaded.

54. The computer-readable medium of claim 51 wherein the processor is multi-threaded.

55. The computer-readable medium of claim 51 wherein the task can only perform idle processing until the event occurs.

56. The computer-readable medium of claim 51 wherein the task can perform no processing until the event occurs.

57. The computer-readable medium of claim 51 wherein the operating system reassigns the processor resource to the task after an external event directed to the task occurs.

58. The computer-readable medium of claim 51 wherein the notification is received by the operating system in response to informing the task that the task is to be swapped out from processor resource utilization.

59. A computer-readable medium for causing clients executing on a computer system to coordinate assignment of a processor resource with an operating system, by:

under control of a task,

determining whether the task cannot productively use the processor resource until an event occurs; and

when the task determines that it cannot productively use the processor resource, notifying the operating system that the task cannot productively use the resource

wherein the operating system unassigns the resource from the task until after an event occurs.

60. The computer-readable medium of claim 59 wherein the processor is single-threaded.

61. The computer-readable medium of claim 59 wherein the processor is multi-threaded.

62. The computer-readable medium of claim 59 wherein the operating system upon notification assigns the processor resource to another task.

63. The computer-readable medium of claim 59 wherein the task can only perform idle processing until the event occurs.

64. The computer-readable medium of claim 59 wherein the task determines whether it cannot productively use the processor resource until an event occurs after receiving an indication from the operating system that the task is to be swapped out from processor utilization.

65. The computer-readable medium of claim 59 wherein the task notifies the operating system after receiving an indication from the operating system that the task is to be swapped out from processor utilization.

66. The computer-readable medium of claim 59 wherein the task can perform no processing until the event occurs.

67. A method in a computer system for placing a task in a known state, the task having multiple threads executing on streams of the computer system, the method comprising:

notifying each of the threads of the task executing on a stream to enter a known state; and

for each of the threads,

in response to receiving the notification, entering the known so that so that an action can be performed with the task being in the known state.

68. The method of claim 67 wherein the known state is a quiescent state.

69. The method of claim 67 wherein the known state is where each of the threads is executing idle instructions.

70. The method of claim 67 wherein the known state is where the threads stop executing instructions.

71. The method of claim 67 wherein the task is assigned to a protection domain and the notifying including raising a domain signal for the protection domain.

72. The method of claim 67 wherein prior to entering the known state each thread saves it state.

73. The method of claim 67 wherein a thread of the task initiates the notifying.

74. The method of claim 73 wherein the thread initiates the notifying by sending a request to an operating system.

75. The method of claim 73 wherein the task notifies an operating system that the task is blocked from further productive use of a resource until an event occurs prior to entering the known state.

76. The method of claim 67 wherein the action to be performed is swapping out the task from processor utilization.

77. The method of claim 76 wherein prior to entering the known state each thread saves its state information.

78. The method of claim 67 wherein each thread of the task has state information and the action to be performed is review of the state information.

79. The method of claim 78 wherein the review of the state information is by a debugger.

80. The method of claim 79 wherein the debugger executes as a thread of the task that does not enter the known state.

81. The method of claim 67 wherein the action is to process signal indicated by an operating system.

82. The method of claim 67 wherein the action is to perform an inter-thread long jump.

83. The method of claim 67 wherein the known state is waiting on a synchronization indication.

84. The method of claim 83 wherein the waiting is performed by accessing a memory location with a synchronization access mode of future.

85. The method of claim 67 wherein entering the known state includes invoking an operating system call.

86. The method of claim 67 wherein a master thread is designated to notifying an operating system when the task is in the known state.

87. The method of claim 67 wherein one of the threads enters a known state of processing signals while the other threads are in a known state that is quiescent state.

88. The method of claim 87 wherein after processing the signals each of the threads exit the known state.

89. A method in a computer system for a task to exit a known state, the computer system supporting multiple streams, the task having multiple threads executing on the streams, the method comprising:

notifying each of the threads of the task to exit the known state; and

for each thread,

in response to receiving the notification, executing instructions that were to be executed prior to entering the known state.

90. The method of claim 89 wherein the known state is a quiescent state.

91. The method of claim 89 wherein the known state is where the threads are executing idle instructions.

92. The method of claim 89 wherein the known state is where the threads stop executing instructions.

93. The method of claim 89 wherein the known state is where the threads are waiting on a synchronization indication.

94. The method of claim 93 wherein the notifying includes indicating the synchronization.

95. The method of claim 89 wherein each thread restores state information that was save prior to the state entering the known state.

96. The method of claim 95 wherein one thread performs signal processing upon exiting the known state.

97. The method of claim 96 wherein the other threads wait until the signals are processed before executing instructions that were to be executed prior to entering the known state.

99. The method of claim 89 including reserving a number of streams for the task.

100. The method of claim 89 wherein after receiving the notification, the task creates streams for the threads.

101. A method in a computer system for assigning a processor resource to a thread of a task, the method comprising:

under control of a thread of the task, invoking an operating system call that will block waiting for the occurrence of an event;

under control of the operating system, invoking a routine of the task so that the routine can assign the processor resource to another thread of the task.

102. The method of claim 101 wherein the processor resource is a stream of a processor that supports multiple streams.

103. The method of claim 101 wherein the task registers the routine with the operating prior to invoking the operating system call.

104. The method of claim 101 including notifying the task when a operating system call completes.

105. The method of claim 101 wherein the computer system is a MTA computer system.